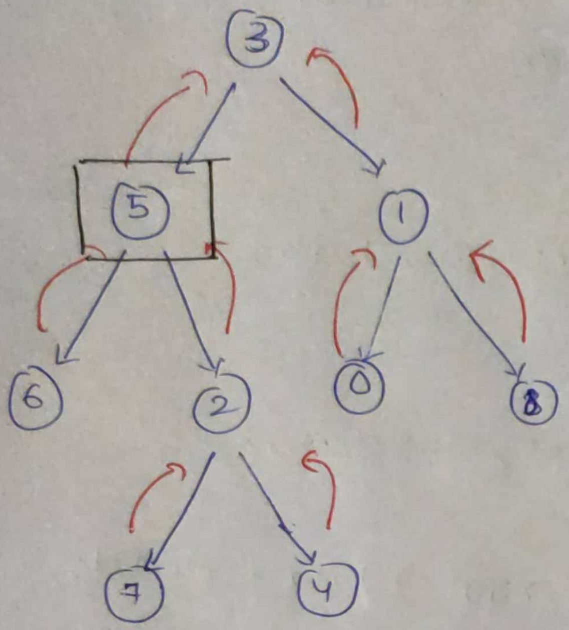
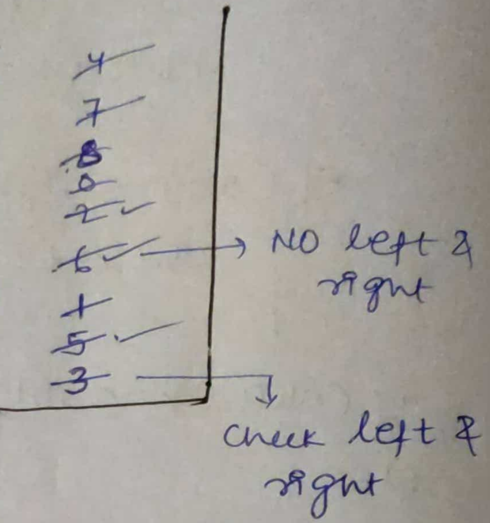
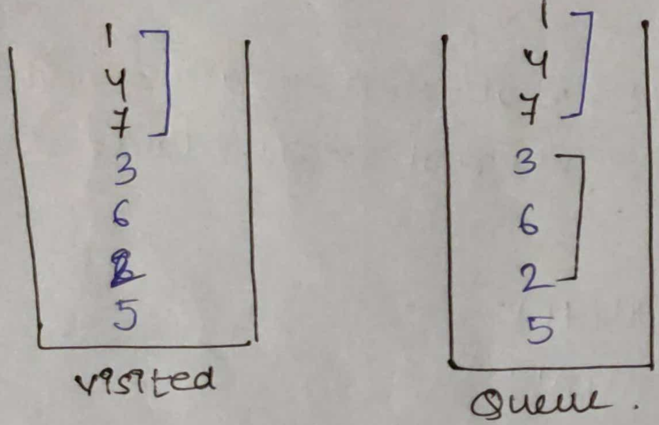
Nodes at a distance K.

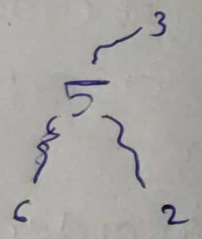
K = 2, target = 5

We need to perform the BFS traversal.









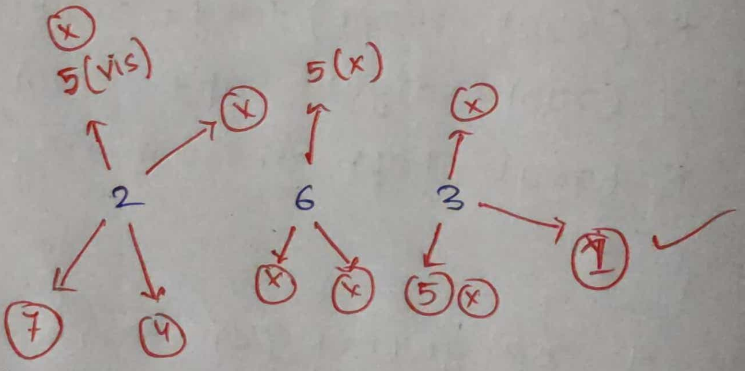
- check from node 5 (the left, right and parent node) & push them into the queue and also in the visited ds.

dis = ~~0~~ ~~1~~ 2

- check (2, 6, 3)

node 3 → node 1

as node is not visited before push.



distances increases by 1, dis → 1 ⇒ dis ⇒ 2.

That's what we wanted

- check parent
- move by 1 dist.

last nodes vis must be pushed into some vector and return.

1	4	7

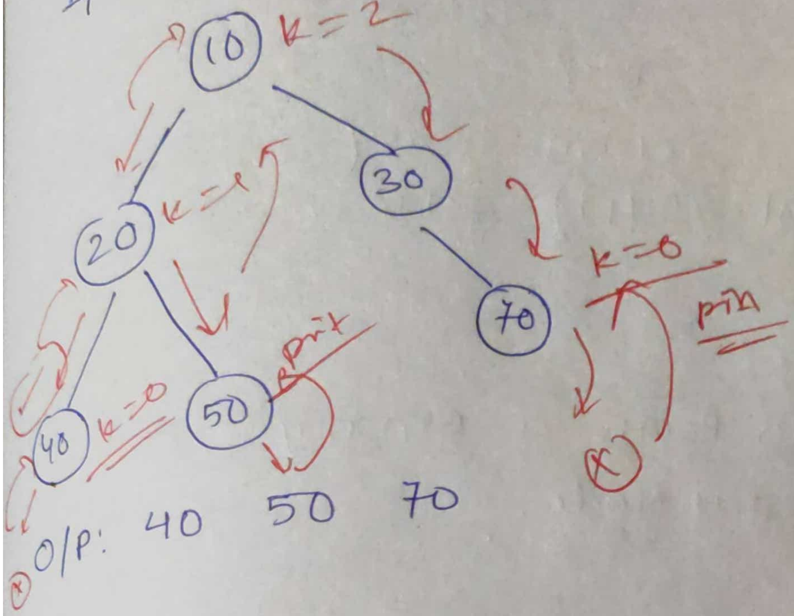
```
void markParents (TreeNode * root, unordered_map <TreeNode*,
                  TreeNode*> & parent_track, TreeNode *target)

{
  queue <TreeNode *> q;
  q.push(root);
  while (!q.empty()) {
    TreeNode * current = q.front();
    q.pop();
    if (curr->left){
      parent_track[current->left] = current;
      q.push (current->left);
    }
    if (current->right) {
      parent_track[current->right] = current;
      q.push (current->right);
    }
  }
}
vector <int> distanceK (TreeNode * root, TreeNode * target,
                        int k) {
unordred_map < TreeNode*, TreeNode *> parent_track;
mark Parents (root, parent_track, target);
unordered_map < TreeNode*, bool> vis;
queue <TreeNode *> q;
q. push (target);
vis[target] = ~~level~~ true;
int curr_lev = 0;
  while(!q.empty()){
    int size = q.size();
    if (curr_lev++  == k) break;
    for(int i=0; i< size; i++){
      TreeNode * current = q.front();
      q.pop();
if (current->left && !vis[current->left]) {
      q.push (current->left);
      vis[current->left] = true;
  }
if (current->right && !vis[current->right]) {
      q.push (current->right);
      vis[current->right] = true;
  }
if (parent_track [current] && !visited [parent_track[current] ]) {
      q. push (parent_track[current]);
      vis [parent_track[current]] = true;
  }
 }
}
 vector<int> result;
 while (!q.empty()) {
     TreeNode * current = q.front();
     q.pop();
     res.push_back (current->val);
  }
 return result~~t~~;
}
```

TC : O(n) + O(n) + O(nlogn)

SC: O(n) + O(n) + O(n)

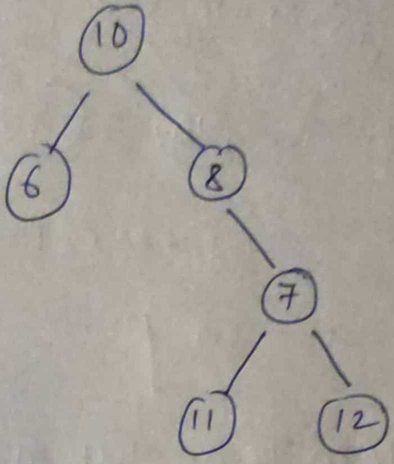
Print Nodes at a distance K from root of BT.

I/P: K = 2



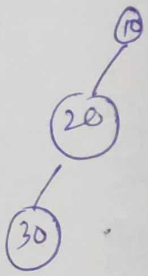
O/P: 40 50 70

K=3,



O/P: 11 12

K=1,

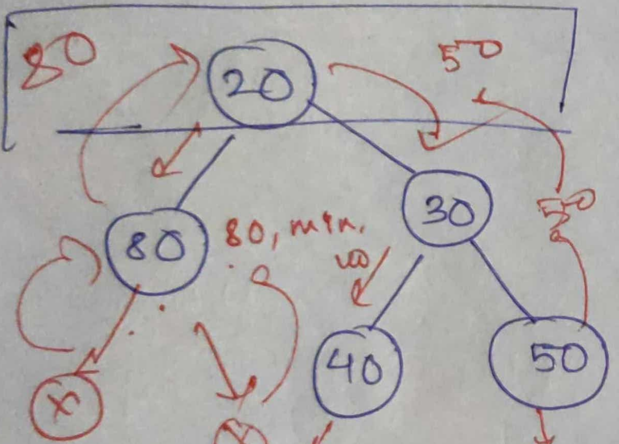


O/P: 20.

Code:-

```
void printKDist(Node *root, int k){
      if (root == NULL) return;
   if (k==0){
      cout << root->key;
   }
  else {
      printKDist(root->left, k-1);
      printKDist(root->right, k-1);
   }
 }
```

Maximum Value Node in Binary Tree :-



```
int getMax (Node * root){
    if (root == NULL)
        return INT_MIN;
    else
    return max (root->key,
(getMax (root->left)), getMax (root->right)
```

Max = 80